

# NeuralSVG: An Implicit Representation for Text-to-Vector Generation

SAGI POLACZEK, Tel Aviv University, Israel

YUVAL ALALUF, Tel Aviv University, Israel

ELAD RICHARDSON, Tel Aviv University, Israel

Yael Vinker, Tel Aviv University, Israel and MIT CSAIL, USA

DANIEL COHEN-OR, Tel Aviv University, Israel

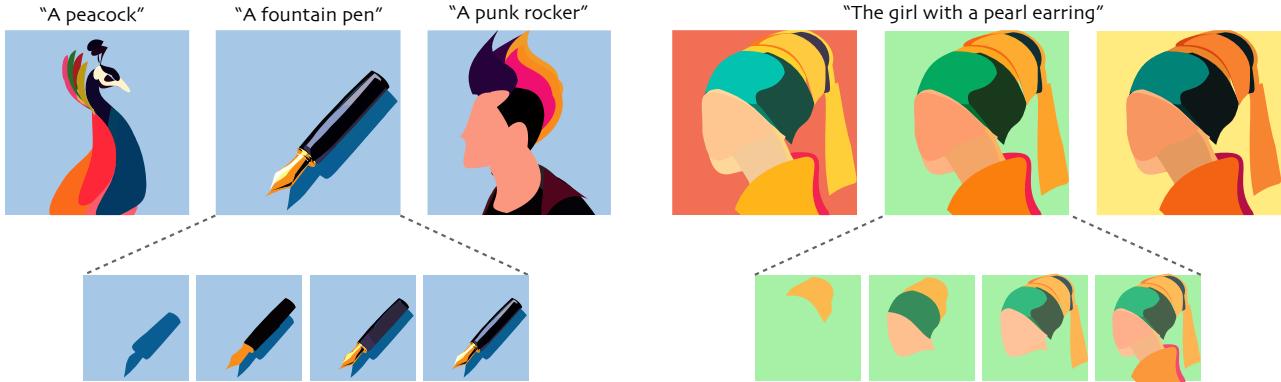


Fig. 1. NeuralSVG generates vector graphics from text prompts with ordered and editable shapes. Our method supports dynamic background color conditioning, facilitating the generation of multiple color palettes for a single learned representation (right).

Vector graphics are essential in design, providing artists with a versatile medium for creating resolution-independent and highly editable visual content. Recent advancements in vision-language and diffusion models have fueled interest in text-to-vector graphics generation. However, existing approaches often suffer from over-parameterized outputs or treat the layered structure – a core feature of vector graphics – as a secondary goal, diminishing their practical use. Recognizing the importance of layered SVG representations, we propose NeuralSVG, an implicit neural representation for generating vector graphics from text prompts. Inspired by Neural Radiance Fields (NeRFs), NeuralSVG encodes the entire scene into the weights of a small MLP network, optimized using Score Distillation Sampling (SDS). To encourage a layered structure in the generated SVG, we introduce a dropout-based regularization technique that strengthens the standalone meaning of each shape. We additionally demonstrate that utilizing a neural representation provides an added benefit of inference-time control, enabling users to dynamically adapt the generated SVG based on user-provided inputs, all with a single learned representation. Through extensive qualitative and quantitative evaluations, we demonstrate that NeuralSVG outperforms existing methods in generating structured and flexible SVG. Project page: <https://sagipolaczek.github.io/NeuralSVG/>.

## 1 INTRODUCTION

Vector graphics represent images using parametric shapes, such as circles, polygons, lines, and curves, in contrast to rasterized images, which rely on pixel-level representations. Unlike raster images, vector graphics are resolution-independent, easily editable, and particularly effective for creating simplified visuals. These advantages make vector graphics a preferred choice in fields such as design, web development, and data visualization. Recent research has sought to automate the generation of vector graphics, aiming to create high-quality, scalable visual content accessible to both experts and non-experts alike.

With recent advancements in large-scale vision-language models [Yin et al. 2024] and image diffusion models [Po et al. 2023], there has been a growing interest in introducing these strong priors to directly generate vector graphics from text prompts [Jain et al. 2023; Thamizharasan et al. 2024; Xing et al. 2024; Zhang et al. 2024]. However, existing methods, while technically producing vector graphics, often result in over-parameterized outputs composed of almost pixel-like shapes, thus losing the original motivation and core advantages of editable vector graphics (see Figure 2).

Notably, the editable nature of SVGs is inherently linked to their layered representation. These layers separate elements like backgrounds, text, and shapes for easier navigation, enable independent editing without affecting other components, and provide a hierarchical structure for stacking and visual clarity. Motivated by this, several works have proposed methods for generating layer-based SVG representations [Thamizharasan et al. 2024; Zhang et al. 2024]. However, these approaches often depend on multiple post-processing stages to construct a meaningful layered structure. Ideally, the SVG generation process itself should account for the hierarchical nature of SVGs, promoting the creation of shapes that possess standalone semantic meaning while contributing to the overall composition.

In this work, we introduce *NeuralSVG*, an implicit neural representation for text-to-vector generation that takes into account the layered structure of vector graphics and offers greater flexibility in the generation process. Inspired by Neural Radiance Fields (NeRFs), which output individual points in space that are then aggregated into a scene, we propose a network that outputs individual shapes which are then aggregated to form the complete SVG. Following prior work, the network weights are optimized using the standard Score Distillation Sampling (SDS) loss [Poole et al. 2022]. In this



**Fig. 2. The Importance of Layers and Compact Shapes in SVGs for Editability.** Left: SVGs are typically composed of ordered layers (e.g., the gray background and trees are placed behind the house) and individual shapes that represent complete components in an editable manner (e.g., snow can be removed or adjusted by modifying a few shapes). Right: An SVG that may appear visually appealing when rendered but lacks practical use for editing or control, as its individual components are difficult to modify.

formulation, the entire network encodes the complete SVG as an implicit neural representation, defined by its learned weights. To promote a semantic and ordered representation, we further introduce a dedicated dropout-based regularization method during the optimization process. This method encourages each learned shape to have a meaningful and ordered role in the overall composition.

Importantly, using a neural representation introduces greater flexibility in utilizing and extending SVGs. Specifically, we demonstrate that our implicit representation enables inference-time control over the generated asset. For instance, by conditioning the generation on a target background color, our network can learn to produce a color palette for the SVG that best complements this background. As illustrated in Figure 1, this enables the creation of dynamic SVGs that adapt to user-specific preferences.

We evaluate NeuralSVG through comprehensive qualitative and quantitative experiments, demonstrating improved performance across a diverse range of inputs compared to existing methods. Notably, we show that our single-stage framework generates meaningful individual shapes, providing users with a well-structured and layered representation. Additionally, we demonstrate that NeuralSVG can be adapted to produce vectorized sketches without any modifications. Finally, as a key distinguishing feature, we highlight how NeuralSVG supports additional user inputs, creating an adaptive SVG representation that can be dynamically adjusted at inference time beyond the capabilities of standard SVG representations.

## 2 RELATED WORK

### 2.1 Vector Representation

Scalable Vector Graphics (SVGs) [Jackson and Northway 2005] offer a flexible and powerful medium for representing visual concepts, leveraging primitives such as Bézier curves [Beziers 1986]. Extensive research has focused on learning neural-based representations of SVGs. SketchRNN [Ha and Eck 2017] uses a recurrent neural network (RNN) to generate vector paths for sketches, while DeepSVG [Carlier et al. 2020] adopts a hierarchical Transformer model to create vector icons with multiple paths. More recently, IconShop [Wu et al. 2023] represents SVGs as token sequences.

### 2.2 Text-to-Image Generation

Recent advancements in large-scale generative models [Po et al. 2023; Yin et al. 2024] have rapidly transformed content creation, especially in visual content generation. Among these, large-scale

diffusion models [Balaji et al. 2023; Ding et al. 2022; Nichol et al. 2021; Ramesh et al. 2022; Rombach et al. 2022; Saharia et al. 2022; Shakhmatov et al. 2022] have achieved unprecedented levels of quality, diversity, and fidelity in their outputs. These models have also spurred the development of various text-guided tasks. Central to this progress is the Score Distillation Sampling (SDS) loss, introduced by Poole et al. [2022], which has proven highly effective for extracting meaningful signals from pretrained text-to-image diffusion models. SDS has enabled a wide range of applications, including text-to-3D generation [Lin et al. 2023; Metzer et al. 2023; Poole et al. 2022; Richardson et al. 2023; Wang et al. 2023a, 2024b], image editing [Hertz et al. 2023; Kim et al. 2025; Koo et al. 2024], sketch generation [Gal et al. 2024; Iluz et al. 2023; Kim et al. 2023; Mo et al. 2024; Xing et al. 2023], and text-to-SVG generation.

### 2.3 Vector Graphics Generation

Early vector graphics generation approaches relied on sequence-based learning applied to vector representations [Carlier et al. 2020; Ganin et al. 2018; Ha and Eck 2017; Lopes et al. 2019; Wang et al. 2023b; Wu et al. 2023], but their dependence on vector datasets limited their generalization to more complex generations. Advances in differentiable rendering [Li et al. 2020; Mihai and Hare 2021; Reddy et al. 2020; Zheng et al. 2018] have enabled vector synthesis using raster-based losses [Ma et al. 2022; Reddy et al. 2021; Shen and Chen 2021; Xing et al. 2023]. Additionally, the emergence of large-scale vision-language models, such as CLIP [Radford et al. 2021], had led to innovative methods for sketch and vector generation [Frans et al. 2022; Jain 2021; Mirowski et al. 2022; Rodriguez et al. 2023; Song et al. 2023; Tian and Ha 2022; Vinker et al. 2023, 2022, 2024].

Recent research has focused on integrating diffusion models into vector graphics generation. A key approach optimizes geometric and color parameters of primitives using diffusion model priors with SDS-based losses [Jain et al. 2023; Thamizharasan et al. 2024; Xing et al. 2024; Zhang et al. 2024]. However, these methods often suffer from redundant and degraded geometry due to the absence of ordering constraints. For instance, SVGDreamer [Xing et al. 2024] requires numerous shapes (e.g.,  $\sim 512$ ) and supports only basic scene decomposition into background and foreground. Text-to-Vector [Zhang et al. 2024] trains a Variational Autoencoder (VAE) to encode valid geometric properties into a path latent space. Their method employs a two-stage path optimization process for text-to-vector generation, utilizing the learned latent space with an SDS-based loss. As a post-processing step, they simplify the obtained paths to produce a layer-wise representation. NIVE $\lambda$  [Thamizharasan et al. 2024] trains an MLP to learn decomposable SVG layers, generating layered outputs in pixel space that are vectorized into Bézier curves via marching squares in post-processing.

Several methods combine text-to-image generation with image vectorization techniques [Hirschorn et al. 2024; Ma et al. 2022; Wang et al. 2024a] to produce vector graphics [Chen et al. 2023; Du et al. 2023; Kopf and Lischinski 2011]. For instance, LIVE [Ma et al. 2022] employs a differentiable rasterizer to iteratively optimize closed Bézier paths. Wang et al. [2024a] combined Score Distillation Sampling and semantic segmentation to iteratively simplify the input image into vectorized layers.

In this work, we propose a novel implicit neural representation for SVGs, encoding the SVG as the weights of a small MLP neural network. This neural representation provides a more interpretable generation process with enhanced user control, allowing customization of parameters such as the number of shapes, background color, and aspect ratio, all within a single network.

## 2.4 Ordered Representations

Ordered representations, such as those obtained through Principal Component Analysis (PCA), where dimensions are ranked by their relative importance, are extensively used in machine learning and statistics. Rippel *et al.* [2014] demonstrated that neural networks could be encouraged to learn ordered representations by applying a specialized form of dropout on hidden units.

In the context of generative models, the exploration of ordered representations is still a developing area. Alaluf *et al.* [2023] utilize ordered representations to personalize text-to-image models, enabling inference-time control over the reconstruction and editability of learned concepts. Zhang *et al.* [2024] introduce a post-processing method for SVGs, where layer-wise structures are extracted from complete SVGs through a path simplification process. In this work, we adopt an ordered-centric approach to SVG generation, integrating the layered structure directly into the generation process.

## 3 PRELIMINARIES

**Score-Distillation Sampling.** Score-Distillation Sampling (SDS), introduced by Poole *et al.* [2022], has emerged as a prominent technique for extracting meaningful signals from pretrained text-to-image diffusion models. The authors demonstrated how the standard diffusion loss can be leveraged to optimize the parameters of a NeRF [Mildenhall *et al.* 2021] model for text-to-3D generation.

Given an image  $x$  (e.g., a radiance field rendered from a specific viewpoint) synthesized by a model with parameters  $\phi$ , the image is noised to an intermediate diffusion timestep  $t$  as follows:

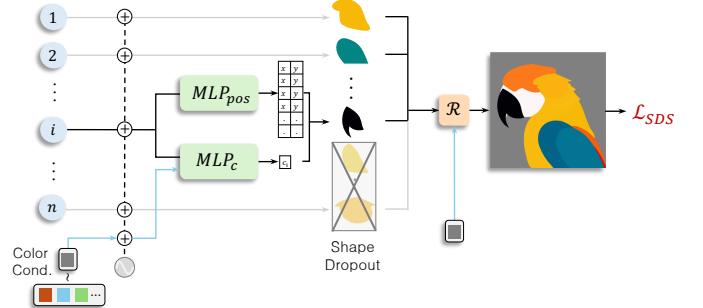
$$x_t = \alpha_t x + \sigma_t \epsilon \quad (1)$$

where  $\epsilon \sim \mathcal{N}(0, 1)$  represents a noise sample, and  $\alpha_t$  and  $\sigma_t$  are parameters defined by the denoising scheduler.

The noised image is then passed through a pretrained, frozen denoising model conditioned on a prompt  $p$ , which aims to predict the added noise  $\epsilon$ . The deviation between the predicted noise and the true added noise serves as a measure of the difference between the input image  $x$  and one that better matches the given prompt. The corresponding gradients can then be used to update the parameters  $\phi$  of the original synthesis model, guiding it to generate outputs more aligned with the prompt. The loss function is given by:

$$\nabla_{\theta} \mathcal{L}_{\text{SDS}} = \mathbb{E}_{t, \epsilon} \left[ w(t) \left( \hat{\epsilon}_{\phi}(St; p, t) - \epsilon \right) \frac{\partial s}{\partial \theta} \right], \quad (2)$$

where  $\hat{\epsilon}_{\phi}$  is the noise predicted by the denoising model, and  $w(t)$  is a weighting function that depends on the diffusion timestep  $t$ . Intuitively, this iterative process progressively aligns the synthesis model with the conditioning prompt  $p$ . Here, we adopt this approach to update the weights of our network representing the SVG scene.



**Fig. 3. NeuralSVG Overview.** Input indices  $\{1, \dots, n\}$ , each corresponding to a single shape, are processed through two parallel branches:  $MLP_{\text{pos}}$ , which predicts the control points of the shape, and  $MLP_{\text{c}}$ , which predicts its RGB color. The predicted shapes and colors are then aggregated and rendered using a differentiable rasterizer  $\mathcal{R}$ . To encourage a meaningful ordering of the shape primitives, a truncation index is randomly sampled during training, and all shapes above this index are dropped. The final rendered vector graphic is optimized to align with the user-provided text prompt using an SDS loss [Poole *et al.* 2022], guided by a trained diffusion model. Additionally, random background colors are sampled during training, with their RGB values passed to  $MLP_{\text{c}}$  and  $\mathcal{R}$ .

## 4 METHOD

Given a user-provided text prompt, NeuralSVG learns an implicit neural representation of the corresponding vector graphics scene. We begin by describing our network architecture and training scheme. We then introduce a dropout-based regularization technique applied during optimization, which is designed to establish a meaningful ordering of the learned shape primitives. Finally, we demonstrate how our neural representation enables greater user flexibility, allowing users to better customize the generated SVGs using a single learned representation. A high-level overview of NeuralSVG is illustrated in Figure 3.

### 4.1 Neural SVG Representation

Our neural SVG representation is inspired by the implicit representation of Neural Radiance Fields (NeRFs) [Mildenhall *et al.* 2021], where 3D pixel coordinates are mapped to spatial points through a compact mapping network. Similarly, we represent an SVG implicitly as a set of indices,  $\{1, 2, \dots, n\}$ , where each index  $i$  corresponds to a single shape  $z_i$  in the SVG. Each shape is defined by four concatenated cubic Bézier curves, with their first and last control points being identical to form a closed shape. This results in 12 control points  $p_i = \{x_j, y_j\}_{j=1}^{12}$  per shape. Each shape is defined by its control points and fill color:  $z_i = (p_i, c_i)$ . Specifically, we learn a function using a small MLP network,  $f_{\theta}$  with learnable weights  $\theta$ :

$$f_{\theta} : i \rightarrow (p_i, c_i), \quad (3)$$

In essence, the MLP takes a shape index  $i \in \{1, 2, \dots, n\}$  as input and outputs the parameters defining the corresponding shape. These individual shapes are aggregated to form the full set of shapes and are then rendered using a differentiable rasterizer [Li *et al.* 2020] to produce the output in pixel space.

In this formulation, the entire vector scene is encoded within the weights of the network. During inference, the network can then be queried to generate the SVG by feeding it with each of the  $n$  indices. Additionally, this neural representation can be extended to accept additional input parameters, such as background color.

## 4.2 Architecture.

Our model consists of three primary components: a positional encoding layer and two Multi-Layer Perceptron (MLP) networks.

**Positional Encoding.** Given the index of the  $i^{\text{th}}$  shape, we first map the scalar value  $i$  to a higher-dimensional space, following prior works on implicit representations [Alaluf et al. 2023; Gal et al. 2024; Mildenhall et al. 2021; Thamizharasan et al. 2024]. Specifically, each input scalar is encoded using Random Fourier Features [Rahimi and Recht 2007; Tancik et al. 2020] into a 128-dimensional vector,  $\gamma(i) \in \mathbb{R}^{128}$ , modulated by 64 random frequencies. The encoding function is defined as:

$$\gamma(i) = [\cos(2\pi Bi) \sin(2\pi Bi)] \quad (4)$$

where  $B$  is a matrix of random frequencies.

**Network Architecture.** Given the high-dimensional encoding of the shape index, we predict the shape’s control parameters and color using an MLP network. To better disentangle the color and shape information, each is predicted using parallel branches. In each branch, the input vector  $v_i = \gamma(i)$  is passed through two fully connected layers, each followed by a LayerNorm [Ba et al. 2016] normalization layer and a LeakyReLU activation. The resulting vector is then passed through a final fully connected layer to produce the outputs  $\hat{p}_i$  or  $\hat{c}_i$ , where  $\hat{p}_i$  is a  $(12 \times 2)$ -dimensional output representing the  $(x, y)$  coordinates of the 12 control points, and  $\hat{c}_i$  is a 3-dimensional output representing the RGB color values:

$$\hat{p}_i = \text{MLP}_{\text{pos}}(v_i) \in \mathbb{R}^{12 \times 2} \quad \hat{c}_i = \text{MLP}_c(v_i) \in [0, 1]^3. \quad (5)$$

Finally, the output color values are additionally passed through a Sigmoid function to ensure the values are between 0 and 1.

## 4.3 Training Scheme

**Initialization.** To calibrate the outputs of the mapping networks for generating points within the rendered canvas, we perform an initialization stage, as is common in text-to-vector approaches [Jain et al. 2023; Thamizharasan et al. 2024; Xing et al. 2024]. Specifically, given the user-provided text prompt  $p$ , we first generate an image using an off-the-shelf text-to-image diffusion model [Rombach et al. 2022]. We then adopt the saliency-based initialization technique proposed by Vinker et al. [2023], identifying salient regions in the image via an attention-based relevancy map. From this map, we sample  $n$  points and convert them into a set of convex shapes with simple geometry. To initialize the corresponding RGB color values, we extract the colors from the relevant pixels in the generated image. This process provides an initial set of  $n$  shape control points  $p_i^{\text{init}}$ , and their corresponding color values,  $c_i^{\text{init}}$ .

Next, we train our network to predict these extracted positions and colors. The network is trained using a simple  $\mathcal{L}_2$  loss to encourage accurate reconstruction of the initialization values:

$$\begin{aligned} \mathcal{L}_{\text{pos}}(i) &= \|\text{MLP}_{\text{pos}}(i) - p_i^{\text{init}}\|_2^2, \\ \mathcal{L}_c(i) &= \|\text{MLP}_c(i) - c_i^{\text{init}}\|_2^2. \end{aligned} \quad (6)$$

Having initialized the outputs of the network, we now turn to describe how the network can be trained to represent the desired vector graphics scene based on the user-provided prompt.

**Training.** To guide the training process, we leverage a pretrained text-to-image diffusion model, specifically Stable Diffusion [Rombach et al. 2022]. To better capture the visual look of SVGs, we fine-tune a LoRA adapter using a small dataset of high-quality vector art images. We provide additional details on this fine-tuning in the supplementary.

Following prior works on text-to-vector generation, the training process is driven by an SDS loss [Poole et al. 2022]. At each training iteration, the full set of  $n$  indices is passed through the network to predict all the control points and colors. These primitives are rendered using the differentiable renderer  $\mathcal{R}$  to produce the current representation of the scene. Finally, we use the SDS loss defined in Equation (2) to update the parameters of our network. Intuitively, the SDS loss guides the network to learn a vector graphics scene that faithfully reflects the desired content specified by the text prompt.

**Encouraging an Ordered Representation.** The above optimization process results in a generated SVG that aligns with the provided prompt. However, it does not inherently promote a layered representation of the scene. Specifically, there is no objective that explicitly encourages a meaningful ordering of shapes, where later shapes build upon earlier ones to enhance the overall composition. Prior works either (1) fail to explicitly address this [Jain et al. 2023; Xing et al. 2024], resulting in unordered shapes being learned, or (2) decompose the SVG in a separate post-processing [Zhang et al. 2024].

To address this, we explicitly encourage an ordered representation to be learned directly *during* the optimization process. Specifically, as illustrated on the right side of Figure 3, we adopt a variant of the Nested Dropout technique [Rippel et al. 2014]. At each iteration, before rendering the current scene, we sample a truncation value  $tr$  and drop all shapes above this value, yielding a simplified scene  $S_{tr}$ :

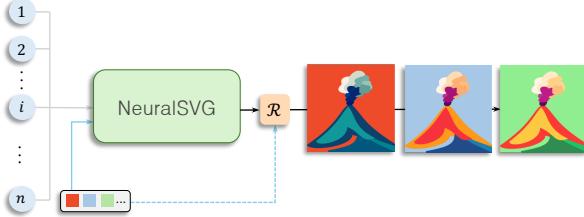
$$\begin{aligned} P_{tr} &= \{p_i\}_{i < tr} & C_{tr} &= \{c_i\}_{i < tr} \\ S_{tr} &= \mathcal{R}(P_{tr}, C_{tr}), \end{aligned} \quad (7)$$

where  $P_{tr}$  and  $C_{tr}$  are the truncated sets of positions and colors.

By randomly dropping shapes during training, the model is encouraged to encode more semantic information into the earlier shapes, which are less likely to be dropped. This technique also provides an additional benefit: enhanced user flexibility at inference. By adjusting the truncation, users can control the number of shapes rendered, tailoring the scene’s complexity to their preferences.

## 4.4 Introducing Additional Controls

Finally, leveraging a neural network to represent SVGs offers the additional benefit of introducing user inputs that can directly control the generated scene, all within a single learned representation.



**Fig. 4. Dynamic color palette control enabled by the NeuralSVG representation.** Given a learned representation of an SVG, users can dynamically adjust the color palette of the SVG by specifying new background colors.

As a motivating example, users can adjust the color palette of the generated SVG by specifying a desired background color, as illustrated in Figure 4. During training, we extend the previously described scheme as follows. At each training step, we sample a background color represented as RGB values. This sampled background color is passed through a positional encoding function and provided as an additional input to the MLP networks, alongside the encoding of the shape index. When rendering, the sampled background color is additionally passed to the renderer to generate the SVG with that background. The sampled colors are chosen either from a set of predefined colors or taken as random RGB values.

At inference time, users can specify any background color to dynamically adjust the color palette of the SVG scene and better match their needs. We illustrate additional controls in Section 5.4.

## 5 EXPERIMENTS

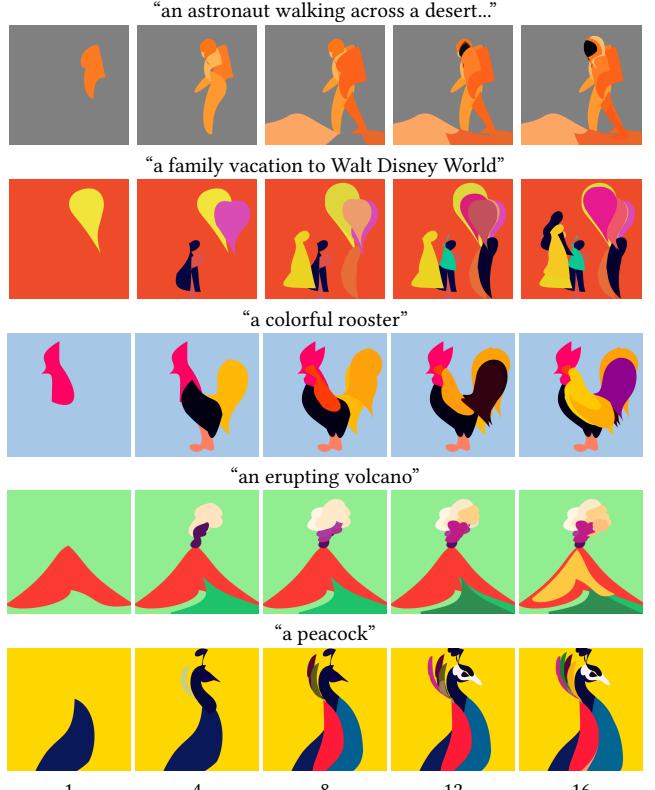
In the following section, we demonstrate the effectiveness of NeuralSVG and the appealing properties of our implicit representation.

**Evaluation Setup.** We evaluate NeuralSVG with respect to state-of-the-art text-to-vector methods including VectorFusion [Jain et al. 2023] initialized using LIVE [Ma et al. 2022], SVGDreamer [Xing et al. 2024], NiVEL [Thamizharasan et al. 2024], and Text-to-Vector [Zhang et al. 2024]. For our evaluations, we use the set of 128 prompts from VectorFusion, as this is the only publicly available text-to-vector prompt evaluation set. For each prompt, we generate five SVGs using five different random seeds. For VectorFusion and SVGDreamer, we evaluate two variants: one using 16 shapes (matching the number of shapes in our method) and another with additional shapes (64 shapes for VectorFusion and 256 for SVGDreamer). We note that SVGDreamer with 512 shapes was not evaluated due to the substantial computational overhead required (over 40GB of VRAM and several hours of runtime on a single A100 GPU).

Furthermore, we note that official implementations for NIVel and Text-to-Vector are unavailable. Our comparison with these methods is based solely on the visual results reported in their respective papers. Finally, unless otherwise specified, we do not apply dropout during inference and output all 16 shapes learned by NeuralSVG.

### 5.1 Qualitative Evaluations and Comparisons

**Qualitative Evaluation.** In Figure 5, we demonstrate text-to-vector results obtained using NeuralSVG. We present outputs generated while retaining a different number of learned shapes in the final rendering: 1, 4, 8, 12, and all 16 shapes. The results show that



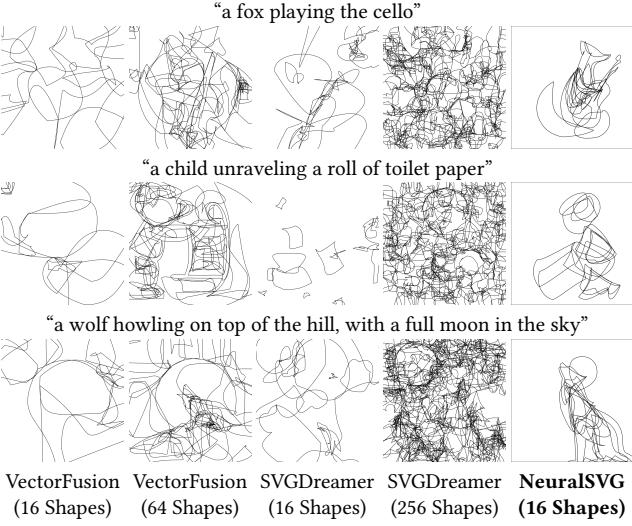
**Fig. 5. Qualitative Results Obtained with NeuralSVG.** We show results generated by our method when keeping a varying number of learned shapes in the final rendering. Even with a small number of shapes (< 4), our approach effectively captures the coarse structure of the scene. Moreover, additional shapes progressively introduce finer details in an ordered manner.

NeuralSVG effectively matches the given prompt even when using only a subset of shapes. Specifically, with just four shapes, the model captures the coarse structure of the scene, such as the outline of the volcano in the fourth row or the body of the peacock in the fifth row. As more shapes are gradually added, the model incorporates finer details in a hierarchical fashion, building upon previously learned shapes. This is most noticeable in the second row where additional people and balloons are gradually added to the complex scene.

**Qualitative Comparisons.** In Figure 6, we compare NeuralSVG to state-of-the-art open-source methods, VectorFusion and SVG-Dreamer. When constrained to the same number of shapes (16), both VectorFusion and SVGDreamer struggle to faithfully represent the desired scene, often missing critical details from the prompt. With increased shape counts – 64 for VectorFusion and 256 for SVG-Dreamer – the methods generate more detailed SVGs that better align with the prompt but exhibit noticeable artifacts. More importantly, both baselines produce uninterpretable and uneditable shapes, limiting their practical usability. We further highlight this redundancy in Figure 7, where we show the outlines of the learned shapes for the results shown here. In contrast, using only 16 shapes, NeuralSVG achieves high-quality results that adhere closely to the prompt, maintain smooth contours, and minimize artifacts, providing users with a more practical result.



**Fig. 6. Qualitative Comparisons.** Visual comparisons to VectorFusion [Jain et al. 2023] and SVGDreamer [Xing et al. 2024] using a varying number of shapes.



**Fig. 7. Shape Outlines of the Generated SVGs.** We present the outlines of SVGs generated by NeuralSVG, VectorFusion, and SVGDreamer. The alternative methods often produce nearly pixel-like shapes that are difficult to modify manually. In contrast, NeuralSVG generates cleaner SVGs, making them more editable and practical.



**Fig. 8. Qualitative Comparisons.** As no code implementations are available, we provide visual comparisons to NiVEL [Thamizharasan et al. 2024] and Text-to-Vector [Zhang et al. 2024] using results shown in their paper.

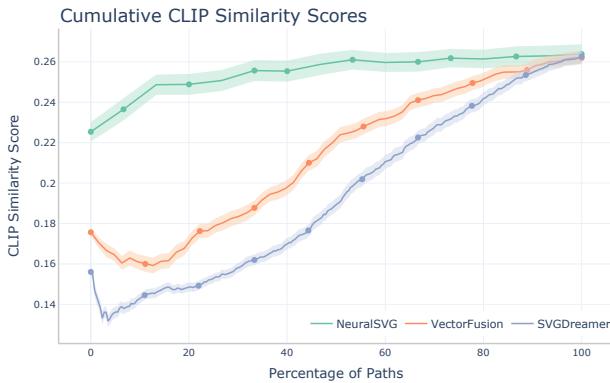
Next, we compare NeuralSVG with more recent but closed-source techniques, as shown in Figure 8. First, when examining the results of NiVEL [Thamizharasan et al. 2024] artifacts are present, particularly along the black contours. This issue arises because NiVEL learns its implicit representation in pixel space and subsequently converts it to an SVG through a post-processing step, which results in pixel-like artifacts. Additionally, a single layer in their implicit representation may encode multiple shapes, leading to potential errors when vectorizing the pixel layers. We observe that the results of Text-to-Vector [Zhang et al. 2024] are comparable to those achieved with NeuralSVG. However, NeuralSVG learns ordered SVGs directly in a single training stage, whereas Text-to-Vector relies on a secondary post-processing step to decompose the SVG into a more editable format. Furthermore, the results presented here are taken directly from their published paper, which restricts our ability to thoroughly analyze the structure of their resulting SVG representations or even know how many shapes were used when rendering.

## 5.2 Quantitative Comparisons

**CLIP-Space Metrics.** To quantitatively evaluate the methods, we follow prior work and employ two CLIP-space metrics. The first metric computes the CLIP-space cosine similarity between the embeddings of the generated SVGs and their corresponding input text prompts. We additionally report the R-Precision (R-Prec), which measures the percentage of generated SVGs that achieve maximal CLIP similarity with their correct prompt among all 128 prompts. We average results across all prompts and five seeds.

**Table 1. CLIP-Based Quantitative Comparisons.** We compute CLIP-space cosine similarities and R-Precision using the CLIP L/14 model on rasterized SVG results, optimized with varying numbers of shapes.

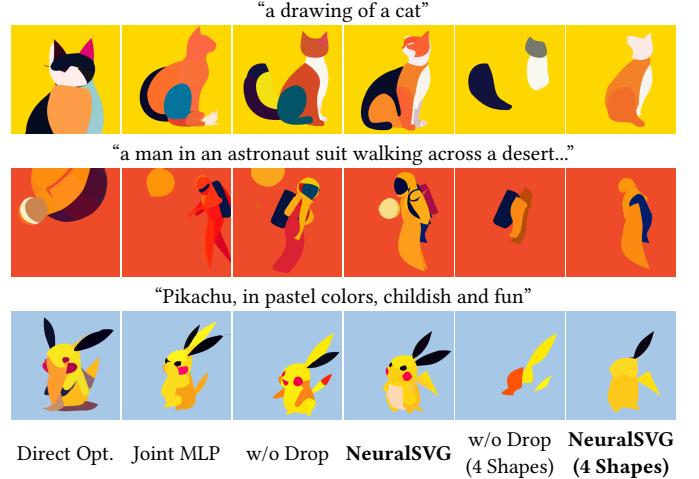
Metric	VectorFusion		SVGDreamer		NeuralSVG
	64	16	256	16	16
R-Precision ↑	83.46	48.03	85.03	43.30	67.18
Text-Image Similarity ↑	26.33	23.47	26.58	20.89	26.94



**Fig. 9. Cumulative CLIP Similarities.** We show CLIP similarities obtained when using a subset of the learned shapes from each method, selected in rendering order. As shown, SVGs produced by NeuralSVG are much more recognizable when using a small percentage of the learned shapes.

Full results are presented in Table 1. When constrained to the same number of shapes, NeuralSVG outperforms both VectorFusion and SVGDreamer across both metrics. This aligns with our visual comparisons, which show that competing methods struggle to generate organized shapes and interpretable scenes under the same constraints. When VectorFusion and SVGDreamer use 64 and 256 shapes, all methods achieve comparable CLIP scores while VectorFusion and SVGDreamer attain higher R-Prec scores than NeuralSVG. However, our visual comparisons reveal that while these higher shape counts improve the image-based metrics, they result in highly disorganized outputs that are impractical for editing. As such, NeuralSVG offers an appealing alternative by generating more organized shapes that create more editable scenes while using a small number of shapes.

**Cumulative CLIP-Space Similarities.** Next, considering the order-centric approach of NeuralSVG, it is important to examine whether the shapes learned by our method align better with CLIP than alternative approaches. To evaluate this, we compute CLIP-space similarities between input text prompts and generated SVGs using a subset of the learned shapes, selected in rendering order. The results in Figure 9 compare NeuralSVG to VectorFusion (64 shapes) and SVGDreamer (256 shapes). As illustrated, SVGs generated by NeuralSVG are significantly more recognizable when using a small fraction of the total shapes. This indicates the early shapes produced by NeuralSVG are semantically meaningful and have a more standalone meaning compared to those generated by alternative methods. Moreover, note that as the total shapes in VectorFusion and SVGDreamer are significantly higher, at 25% and 6%, they already match the shape count used by our full method.



**Fig. 10. Ablation Study.** We validate our key design choices: directly optimizing the shape primitives, using a single MLP network to learn both control point positions and colors, and omitting our ordered dropout technique. The two rightmost columns illustrate results from NeuralSVG trained with and without dropout when rendering the first four learned shapes.

### 5.3 Ablation Studies

Finally, we validate our key design choices, specifically the use of our dropout technique and the two MLP branches. Visual comparisons are presented in Figure 10. First, when attempting to directly optimize the shape primitives, the resulting SVGs often converge to non-smooth shapes and may fail to accurately adhere to the input prompt. This aligns with prior works that observe optimizing parameters via a neural network may assist in attaining smoother and more coherent results [Gal et al. 2024; Vinken et al. 2023]. Next, using a single MLP to predict both the control point positions and colors leads to suboptimal results. For instance, in the second row, the astronaut is incorrectly colored the same as the background while in the first row, the cat appears almost entirely orange, lacking details such as its facial features. Finally, when dropout is omitted, the visual results are comparable to those of our full method, as is expected. However, as illustrated in the two rightmost columns, the learned shapes lack semantic meaning. As a result, when using a small number of shapes, the resulting SVGs are also not easily recognizable by CLIP (see Figure 11). In contrast, NeuralSVG effectively captures the coarse structure of the scene even with a limited number of shapes thanks to our learned ordering.

### 5.4 Additional Controls

**Color Palette Control.** In Figure 21, we demonstrate our method’s ability to dynamically adapt the color palette of the SVG using a single learned representation. Specifically, we show results obtained with colors unobserved during training, illustrating our ability to generalize to new palettes. This flexibility allows users to customize results based on personal preferences at inference, without requiring a dedicated optimization process for each modification.

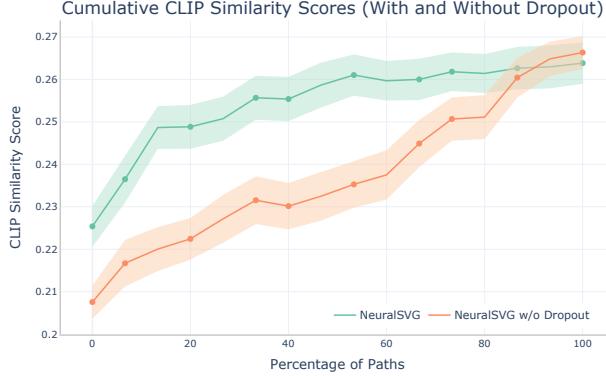


Fig. 11. **Cumulative CLIP Similarities With and Without Dropout.** We show cumulative CLIP similarities achieved by NeuralSVG trained with and without dropout across 50 prompts, using 16 learnable shapes. Consistent Figure 9, our dropout technique improves the recognizability of SVGs.

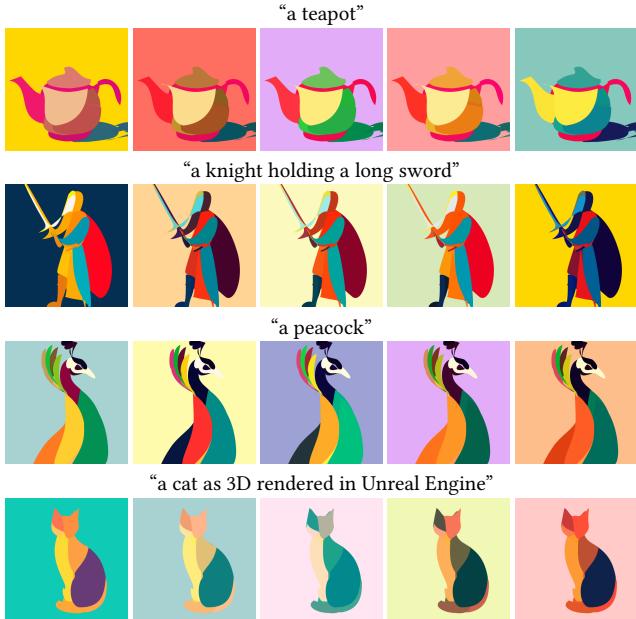


Fig. 12. **Controlling the Color Palette.** Given a learned representation, we render the result using different background colors specified by the user, resulting in varying color palettes in the resulting SVGs.

**Aspect Ratio Control.** Another desired property for controlling SVGs at inference time is easily modifying their target aspect ratio. While one can technically modify the aspect ratio of the SVG manually, successfully generating a pleasing result for a target ratio can still be challenging. We show that by passing an encoding of the desired aspect ratio (e.g., 1:1 or 1:4) to our network and rendering accordingly, our method successfully learns to adapt the same SVG shapes to multiple aspect ratios in the same learned representation. We illustrate this in Figure 13, showing results obtained using aspect ratios of 1:1 and 4:1 when compared to the result one would achieve by automatically “squeezing” the 1:1 result.

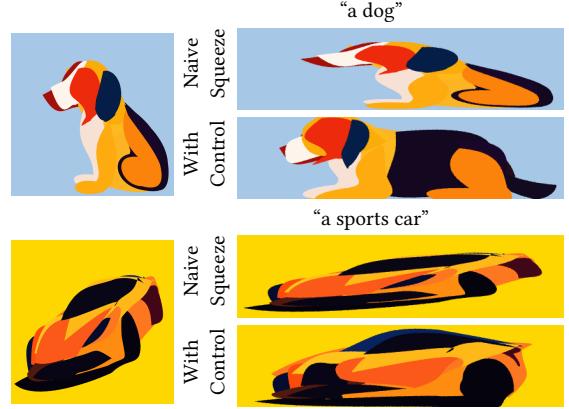


Fig. 13. **Controlling the Aspect Ratio.** We present results from optimizing NeuralSVG with aspect ratios of 1:1 and 4:1. In each pair, the top row shows the naive approach of squeezing the 1:1 output into a 4:1 ratio. The bottom row shows results where the trained network directly outputs the 4:1 ratio.

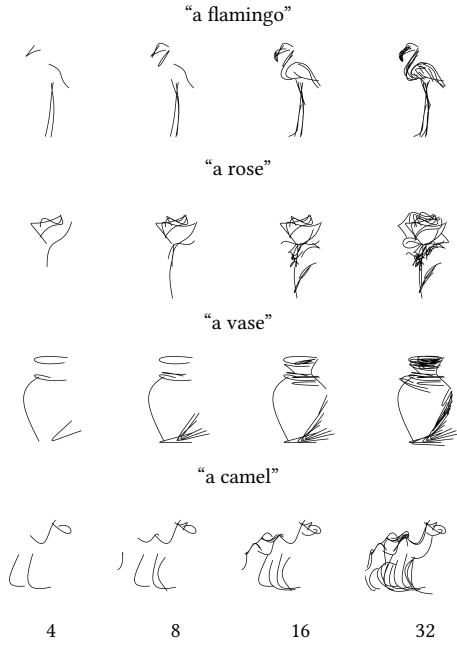


Fig. 14. **Sketch Generation.** NeuralSVG can generate sketches with varying numbers of strokes using a single network, without requiring modifications to our framework.

## 5.5 Sketch Generation

Our approach can also be applied to text-driven sketch generation, generating sketches with ordered strokes. As demonstrated in Figure 14, the first strokes in the sketch depict the desired concept well, while adding more strokes adds details to the sketch. Notably other methods such as NIVeL [Thamizharasan et al. 2024] that use the pixel space as an intermediate stage during training, cannot enforce such stroke-based outputs. In contrast, our approach simply requires modifying the rendering parameters from closed shapes to open shapes when learning the representation.

## 6 CONCLUSION

We introduce NeuralSVG, a novel approach for generating vector graphics directly from text prompts while encouraging a layered structure essential for practical usability. NeuralSVG employs an implicit neural representation to encode the entire SVG within a compact network, optimized using Score Distillation Sampling (SDS). To address a key limitation of existing methods, our approach incorporates a dropout-based regularization technique, promoting the creation of semantically meaningful and well-ordered shapes. In addition to producing structured outputs, NeuralSVG offers enhanced inference-time control, enabling users to adapt the generated SVGs to their preferences, such as adjusting the color palette. We hope this work encourages further exploration into learning meaningful neural representations for vector graphics that are both practical for real-world design applications and provide users with greater flexibility through a more general representation.

## REFERENCES

- Yuval Alaluf, Elad Richardson, Gal Metzger, and Daniel Cohen-Or. 2023. A neural space-time representation for text-to-image personalization. *ACM Transactions on Graphics (TOG)* 42, 6 (2023), 1–10.
- Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. arXiv:1607.06450 [stat.ML] <https://arxiv.org/abs/1607.06450>
- Yogesh Balaji, Seungjün Nah, Xun Huang, Arash Vahdat, Jiaming Song, Qinsheng Zhang, Karsten Kreis, Miika Aittala, Timo Aila, Samuli Laine, Bryan Catanzaro, Tero Karras, and Ming-Yu Liu. 2023. eDiff-L: Text-to-Image Diffusion Models with an Ensemble of Expert Denoisers. arXiv:2211.01324 [cs.CV]
- Pierre Bezier. 1986. Courbes et surfaces, Mathématiques et CAO, 4. *Hermès, Paris* (1986).
- Alexandre Carlier, Martin Danelljan, Alexandre Alahi, and Radu Timofte. 2020. Deepsvg: A hierarchical generative network for vector graphics animation. *Advances in Neural Information Processing Systems* 33 (2020), 16351–16361.
- Ye Chen, Bingbing Ni, Xuanhong Chen, and Zhangli Hu. 2023. Editable image geometric abstraction via neural primitive assembly. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 23514–23523.
- Ming Ding, Wendi Zheng, Wenyi Hong, and Jie Tang. 2022. Cogview2: Faster and better text-to-image generation via hierarchical transformers. *Advances in Neural Information Processing Systems* 35 (2022), 16890–16902.
- Zheng-Jun Du, Liang-Fu Kang, Jianchao Tan, Yotam Gingold, and Kun Xu. 2023. Image vectorization and editing via linear gradient layer decomposition. *ACM Transactions on Graphics (TOG)* 42, 4 (2023), 1–13.
- Kevin Frans, Lisa Soros, and Olaf Witkowski. 2022. Clipdraw: Exploring text-to-drawing synthesis through language-image encoders. *Advances in Neural Information Processing Systems* 35 (2022), 5207–5218.
- Rinon Gal, Yael Vinker, Yuval Alaluf, Amit Bermano, Daniel Cohen-Or, Ariel Shamir, and Gal Chechik. 2024. Breathing Life Into Sketches Using Text-to-Video Priors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 4325–4336.
- Yaroslav Ganin, Tejas Kulkarni, Igor Babuschkin, SM Ali Eslami, and Oriol Vinyals. 2018. Synthesizing programs for images using reinforced adversarial learning. In *International Conference on Machine Learning*. PMLR, 1666–1675.
- David Ha and Douglas Eck. 2017. A neural representation of sketch drawings. *arXiv preprint arXiv:1704.03477* (2017).
- Amir Hertz, Kfir Aberman, and Daniel Cohen-Or. 2023. Delta denoising score. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2328–2337.
- Or Hirschorn, Amir Jevnisek, and Shai Avidan. 2024. Optimize & Reduce: A Top-Down Approach for Image Vectorization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 2148–2156.
- Shir Iluz, Yael Vinker, Amir Hertz, Daniel Berio, Daniel Cohen-Or, and Ariel Shamir. 2023. Word-as-image for semantic typography. *ACM Transactions on Graphics (TOG)* 42, 4 (2023), 1–11.
- Dean Jackson and Craig Northway. 2005. Scalable vector graphics (svg) full 1.2 specification. *World Wide Web Consortium, Working Draft WD-SVG12-20050413 2* (2005).
- Ajay Jain. 2021. *VectorAscent: Generate vector graphics from a textual description*. <https://github.com/ajayjain/VectorAscent>
- Ajay Jain, Amber Xie, and Pieter Abbeel. 2023. Vectorfusion: Text-to-svg by abstracting pixel-based diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1911–1920.
- Jeongsol Kim, Geon Yeong Park, and Jong Chul Ye. 2025. Dreamsampler: Unifying diffusion sampling and score distillation for image manipulation. In *European Conference on Computer Vision*. Springer, 398–414.
- Subin Kim, Kyungmin Lee, June Suk Choi, Jongheon Jeong, Kihyuk Sohn, and Jinwoo Shin. 2023. Collaborative score distillation for consistent visual editing. *Advances in Neural Information Processing Systems* 36 (2023), 73232–73257.
- Jui Koo, Chanhye Park, and Minhyuk Sung. 2024. Posterior distillation sampling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 13352–13361.
- Johannes Kopf and Dani Lischinski. 2011. Depixelizing pixel art. In *ACM SIGGRAPH 2011 papers*. 1–8.
- Tzu-Mao Li, Michal Lukáć, Michaël Gharbi, and Jonathan Ragan-Kelley. 2020. Differentiable vector graphics rasterization for editing and learning. *ACM Transactions on Graphics (TOG)* 39, 6 (2020), 1–15.
- Chen-Hsuan Lin, Jun Gao, Luming Tang, Towaki Takikawa, Xiaohui Zeng, Xun Huang, Karsten Kreis, Sanja Fidler, Ming-Yu Liu, and Tsung-Yi Lin. 2023. Magic3d: High-resolution text-to-3d content creation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 300–309.
- Raphael Gontijo Lopes, David Ha, Douglas Eck, and Jonathon Shlens. 2019. A learned representation for scalable vector graphics. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 7930–7939.
- Xu Ma, Yuqian Zhou, Xingqian Xu, Bin Sun, Valerii Filev, Nikita Orlov, Yun Fu, and Humphrey Shi. 2022. Towards layer-wise image vectorization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 16314–16323.
- Gal Metzger, Elad Richardson, Or Patashnik, Raja Giryes, and Daniel Cohen-Or. 2023. Latent-nerf for shape-guided generation of 3d shapes and textures. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12663–12673.
- Daniela Mihai and Jonathon Hare. 2021. Differentiable drawing and sketching. *arXiv preprint arXiv:2103.16194* (2021).
- Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. 2021. Nerf: Representing scenes as neural radiance fields for view synthesis. *Commun. ACM* 65, 1 (2021), 99–106.
- Piotr Mirowski, Dylan Banarse, Mateusz Malinowski, Simon Osindero, and Chrisantha Fernando. 2022. Clip-clop: Clip-guided collage and photomontage. *arXiv preprint arXiv:2205.03146* (2022).
- Haoran Mo, Xusheng Lin, Chengying Gao, and Ruomei Wang. 2024. Text-based Vector Sketch Editing with Image Editing Diffusion Prior. In *2024 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 1–6.
- Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. 2021. Glide: Towards photorealistic image generation and editing with text-guided diffusion models. *arXiv preprint arXiv:2112.10741* (2021).
- Ryan Po, Wang Yifan, Vladislav Golyanik, Kfir Aberman, Jonathan T. Barron, Amit H. Bermano, Eric Ryan Chan, Tali Dekel, Aleksander Holynski, Angjoo Kanazawa, C. Karen Liu, Lingjie Liu, Ben Mildenhall, Matthias Nießner, Björn Ommer, Christian Theobalt, Peter Wonka, and Gordon Wetzstein. 2023. State of the Art on Diffusion Models for Visual Computing. arXiv:2310.07204 [cs.AI]
- Ben Poole, Ajay Jain, Jonathan T Barron, and Ben Mildenhall. 2022. Dreamfusion: Text-to-3d using 2d diffusion. *arXiv preprint arXiv:2209.14988* (2022).
- Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *International conference on machine learning*. PMLR, 8748–8763.
- Ali Rahimi and Benjamin Recht. 2007. Random Features for Large-Scale Kernel Machines. In *Advances in Neural Information Processing Systems*, J. Platt, D. Koller, Y. Singer, and S. Roweis (Eds.), Vol. 20. Curran Associates, Inc. [https://proceedings.neurips.cc/paper\\_files/paper/2007/file/013a006f03dbc5392effeb8f18fda755-Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/2007/file/013a006f03dbc5392effeb8f18fda755-Paper.pdf)
- Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. 2022. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125* (2022).
- Pradyumna Reddy, Michael Gharbi, Michal Lukac, and Niloy J Mitra. 2021. Im2vec: Synthesizing vector graphics without vector supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 7342–7351.
- Pradyumna Reddy, Paul Guerrero, Matt Fisher, Wilmot Li, and Niloy J Mitra. 2020. Discovering pattern structure using differentiable compositing. *ACM Transactions on Graphics (TOG)* 39, 6 (2020), 1–15.
- Elad Richardson, Gal Metzger, Yuval Alaluf, Raja Giryes, and Daniel Cohen-Or. 2023. Texture: Text-guided texturing of 3d shapes. In *ACM SIGGRAPH 2023 conference proceedings*. 1–11.
- Oren Rippel, Michael A. Gelbart, and Ryan P. Adams. 2014. Learning Ordered Representations with Nested Dropout. arXiv:1402.0915 [stat.ML] <https://arxiv.org/abs/1402.0915>
- Juan A Rodriguez, Shubham Agarwal, Issam H Laradji, Pau Rodriguez, David Vazquez, Christopher Pal, and Marco Pedersoli. 2023. Starvector: Generating scalable vector graphics code from images. *arXiv preprint arXiv:2312.11556* (2023).

- Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2022. High-resolution image synthesis with latent diffusion models. , 10684–10695 pages.
- Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamayr Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, et al. 2022. Photorealistic text-to-image diffusion models with deep language understanding. *Advances in Neural Information Processing Systems* 35 (2022), 36479–36494.
- Arseniy Shakhmatov, Anton Razhigaev, Aleksandr Nikolich, Vladimir Arkhipkin, Igor Pavlov, Andrey Kuznetsov, and Denis Dimitrov. 2022. Kandinsky 2. <https://github.com/ai-forever/Kandinsky-2>.
- I-Chao Shen and Bing-Yu Chen. 2021. Clipgen: A deep generative model for clipart vectorization and synthesis. *IEEE Transactions on Visualization and Computer Graphics* 28, 12 (2021), 4211–4224.
- Yiren Song, Xuning Shao, Kang Chen, Weidong Zhang, Zhongliang Jing, and Minzhe Li. 2023. Clipvg: Text-guided image manipulation using differentiable vector graphics. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 2312–2320.
- Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. 2020. Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains. arXiv:2006.10739 [cs.CV] <https://arxiv.org/abs/2006.10739>
- Vikas Thamizharasan, Difan Liu, Matthew Fisher, Nanxuan Zhao, Evangelos Kalogerakis, and Michal Lukac. 2024. NIVeL: Neural Implicit Vector Layers for Text-to-Vector Generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 4589–4597.
- Yingtao Tian and David Ha. 2022. Modern evolution strategies for creativity: Fitting concrete images and abstract concepts. In *International conference on computational intelligence in music, sound, art and design (part of evostar)*. Springer. 275–291.
- Yael Vinker, Yuval Alaluf, Daniel Cohen-Or, and Ariel Shamir. 2023. Clipascene: Scene sketching with different types and levels of abstraction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 4146–4156.
- Yael Vinker, Ehsan Pajouheshgar, Jessica Y Bo, Roman Christian Bachmann, Amit Haim Bermano, Daniel Cohen-Or, Amir Zamir, and Ariel Shamir. 2022. Clipasso: Semantically-aware object sketching. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 1–11.
- Yael Vinker, Tamar Rott Shaham, Kristine Zheng, Alex Zhao, Judith E Fan, and Antonio Torralba. 2024. SketchAgent: Language-Driven Sequential Sketch Generation. *arXiv preprint arXiv:2411.17673* (2024).
- Patrick von Platen, Suraj Patil, Anton Lozhkov, Pedro Cuenca, Nathan Lambert, Kashif Rasul, Mishig Daavaadorj, Dhruv Nair, Sayak Paul, William Berman, Yiyi Xu, Steven Liu, and Thomas Wolf. 2022. Diffusers: State-of-the-art diffusion models. <https://github.com/huggingface/diffusers>.
- Haochen Wang, Xiaodan Du, Jiahao Li, Raymond A Yeh, and Greg Shakhnarovich. 2023a. Score jacobian chaining: Lifting pretrained 2d diffusion models for 3d generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12619–12629.
- Yuqing Wang, Yizhi Wang, Longhui Yu, Yuesheng Zhu, and Zhouhui Lian. 2023b. Deepvecfont-v2: Exploiting transformers to synthesize vector fonts with higher quality. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 18320–18328.
- Zhenyu Wang, Jianxi Huang, Zhida Sun, Daniel Cohen-Or, and Min Lu. 2024a. Layered Image Vectorization via Semantic Simplification. *arXiv preprint arXiv:2406.05404* (2024).
- Zhengyi Wang, Cheng Lu, Yikai Wang, Fan Bao, Chongxuan Li, Hang Su, and Jun Zhu. 2024b. Prolificdreamer: High-fidelity and diverse text-to-3d generation with variational score distillation. *Advances in Neural Information Processing Systems* 36 (2024).
- Ronghuan Wu, Wanchao Su, Kede Ma, and Jing Liao. 2023. IconShop: Text-Guided Vector Icon Synthesis with Autoregressive Transformers. *ACM Transactions on Graphics (TOG)* 42, 6 (2023), 1–14.
- Ximing Xing, Chuang Wang, Haitao Zhou, Jing Zhang, Qian Yu, and Dong Xu. 2023. Diffsketcher: Text guided vector sketch synthesis through latent diffusion models. *Advances in Neural Information Processing Systems* 36 (2023), 15869–15889.
- Ximing Xing, Haitao Zhou, Chuang Wang, Jing Zhang, Dong Xu, and Qian Yu. 2024. SVGDreamer: Text guided SVG generation with diffusion model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 4546–4555.
- Shukang Yin, Chaoyou Fu, Sirui Zhao, Ke Li, Xing Sun, Tong Xu, and Enhong Chen. 2024. A Survey on Multimodal Large Language Models. arXiv:2306.13549 [cs.CV]
- Peiying Zhang, Nanxuan Zhao, and Jing Liao. 2024. Text-to-Vector Generation with Neural Path Representation. *ACM Transactions on Graphics (TOG)* 43, 4 (2024), 1–13.
- Ningyuan Zheng, Yifan Jiang, and Dingjiang Huang. 2018. Strokenet: A neural painting environment. In *International Conference on Learning Representations*.

## Appendix

### 7 ADDITIONAL DETAILS

**Training Scheme.** In the pretraining stage, we train the network for up to 300 steps using a constant learning rate of 0.01. For the full training process, we train for 4000 steps, employing a learning rate scheduler that features a linear warm-up from 0 to 0.018, followed by a cosine decay to a final value of 0.012. To improve training stability, we clip the gradients using a maximum norm of 0.1.

For computing the SDS loss, we utilize the Stable Diffusion 2.1 model from the diffusers library [von Platen et al. 2022].

In all experiments, prompts are structured using the following format:

"A minimalist vector art of [object], isolated on a [color] background."

Here, [object] specifies the desired scene to be generated, and [color] represents the background color, which is either sampled during training or provided by the user at inference.

When applying our dropout technique, the indices are sampled as follows: with a probability of 0.7, all 16 shapes are rendered. Otherwise, the truncation index, between 1 and 16, is sampled from an exponential distribution with a temperature value of 3.

**LoRA Fine-Tuning.** As detailed in the main paper, our SDS loss is applied with a LoRA adapter applied to Stable Diffusion 2.1. This adapter was pretrained on a high-quality dataset of vector art images. Specifically, the adapter was trained using 1,600 images spanning 145 different prompts, with minor variations between prompts (e.g., with different background colors). These images were generated using the Simple Vector Flux LoRA (see [renderartist/simplevectorflux](#) from diffusers.).

The LoRA adapter was trained for 15,000 steps with a rank of 4.

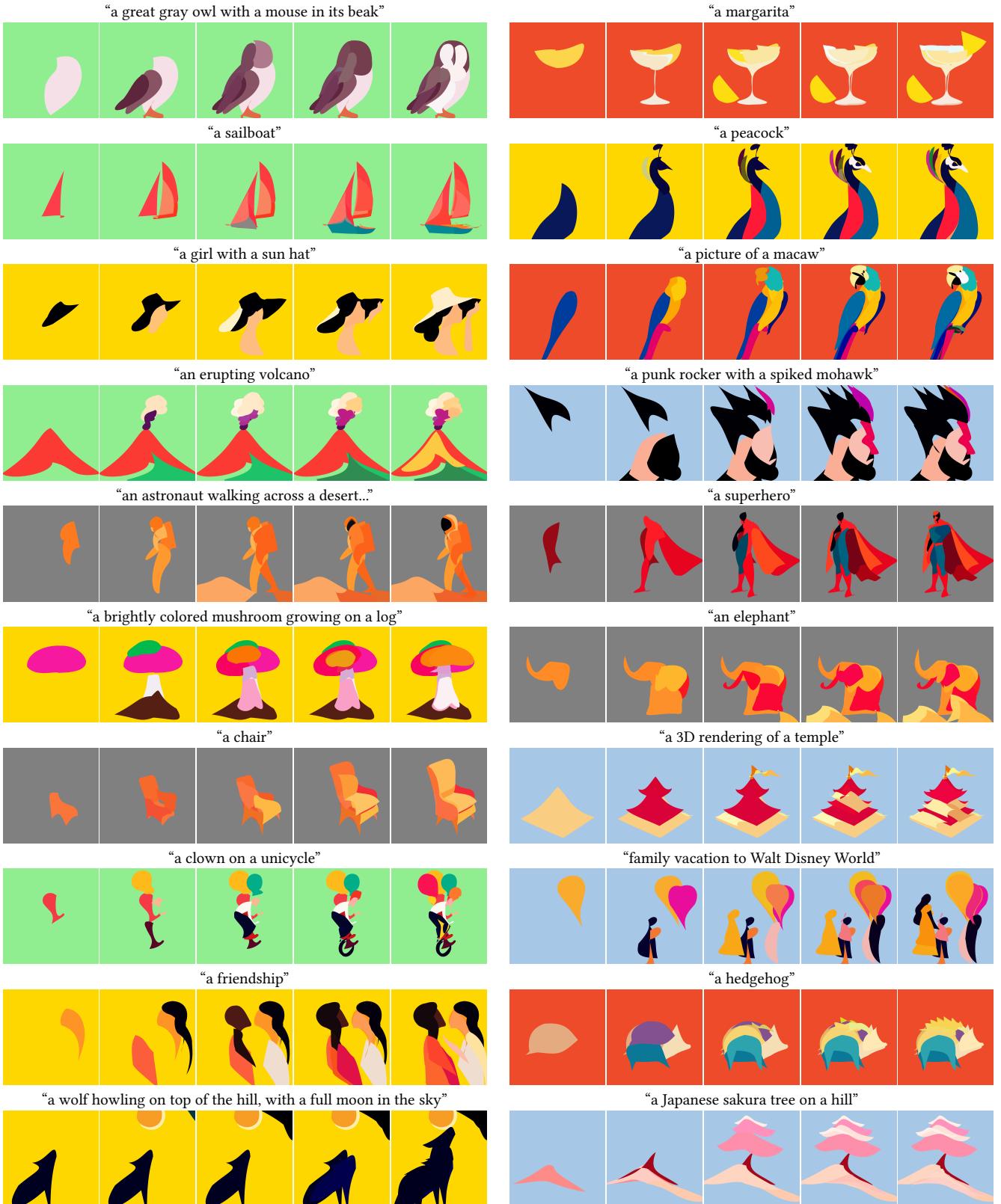
### 8 ADDITIONAL RESULTS AND COMPARISONS

Below, we provide additional qualitative results and comparisons, as follows:

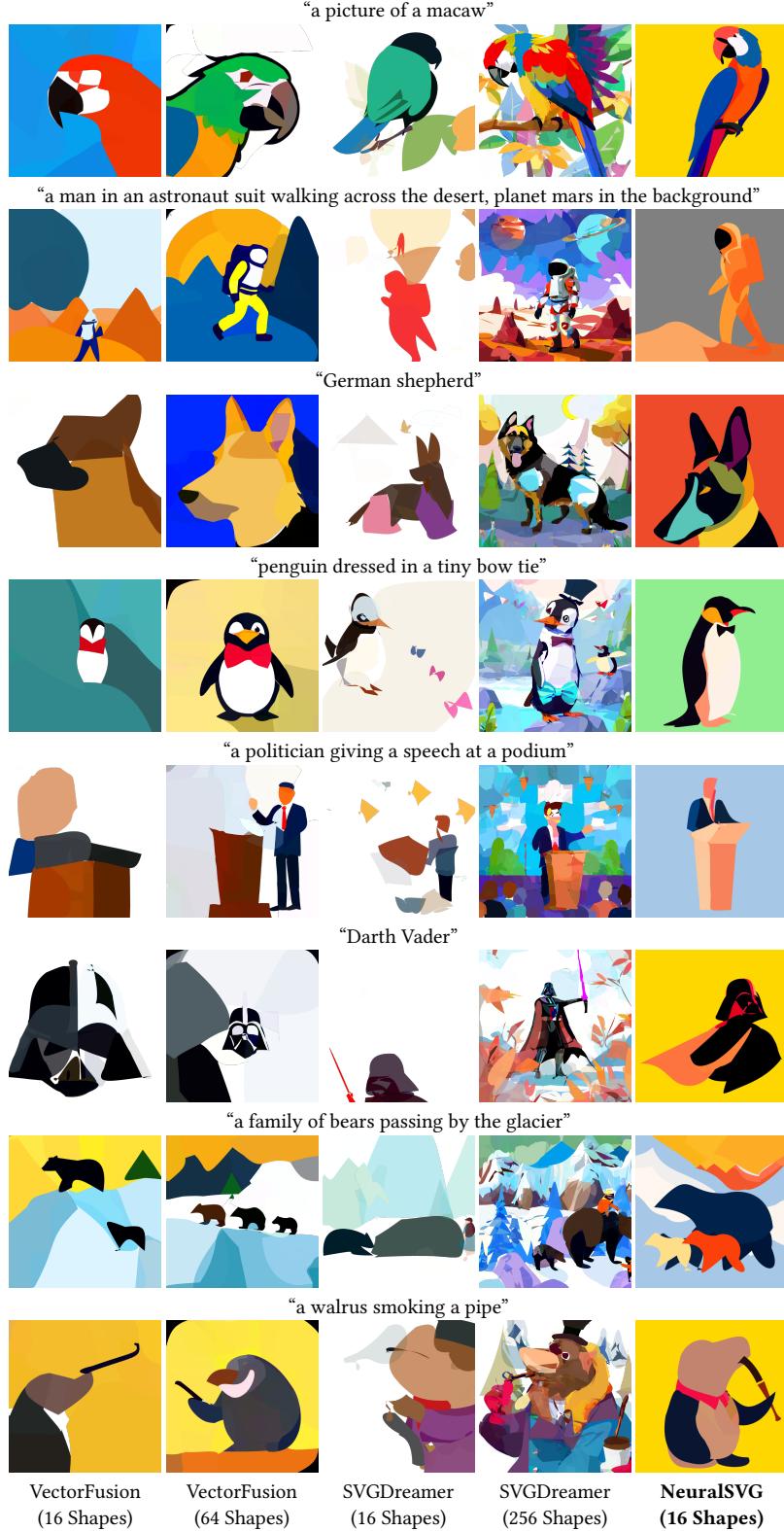
- (1) In Figures 15 and 16, we present additional qualitative results produced by NeuralSVG when applying our dropout technique during inference. Specifically, we vary the number of learned shapes included in the final rendering, showing results with 1, 4, 8, 12, and all 16 shapes.
- (2) In Figures 17 and 18, we provide additional qualitative comparisons to open-source text-to-vector methods VectorFusion [Jain et al. 2023] and SVGDreamer [Xing et al. 2024].
- (3) Following Figure 17, we provide corresponding outlines for the generated SVGs, showing that alternative methods have a tendency to produce nearly pixel-like shapes that are difficult to modify manually while NeuralSVG promotes individual shapes with more semantic meaning and order.
- (4) In Figure 20, we provide additional qualitative comparisons to closed-source techniques NIVeL [Thamizharasan et al. 2024] and Text-to-Vector [Zhang et al. 2024] using results presented in their respective papers.
- (5) Next, in Figures 21 and 22, we show results obtained when rendering the learned SVG with different background colors at inference time, with both seen and unseen colors.
- (6) In Figure 23, we show additional results using our aspect ratio control, allowing us to generate SVGs at different aspect ratios using a single learned representation.
- (7) Finally, in Figure 24, we show sketch generation results obtained using our NeuralSVG framework. Sketches are rendered using a varying number of strokes by modifying the truncation index at inference time. This approach enables a single learned representation to generate sketches at multiple levels of abstraction without modifying our text-to-vector framework.



**Fig. 15. Additional Qualitative Results Obtained with NeuralSVG.** We show results generated by our method when keeping a varying number of learned shapes in the final rendering.



**Fig. 16. Additional Qualitative Results Obtained with NeuralSVG.** We show results generated by our method when keeping a varying number of learned shapes in the final rendering.



**Fig. 17. Additional Qualitative Comparisons.** We provide additional visual comparisons to VectorFusion [Jain et al. 2023] and SVGDreamer [Xing et al. 2024] using a varying number of shapes.



**Fig. 18. Additional Qualitative Comparisons.** We provide additional visual comparisons to VectorFusion [Jain et al. 2023] and SVGDreamer [Xing et al. 2024] using a varying number of shapes.



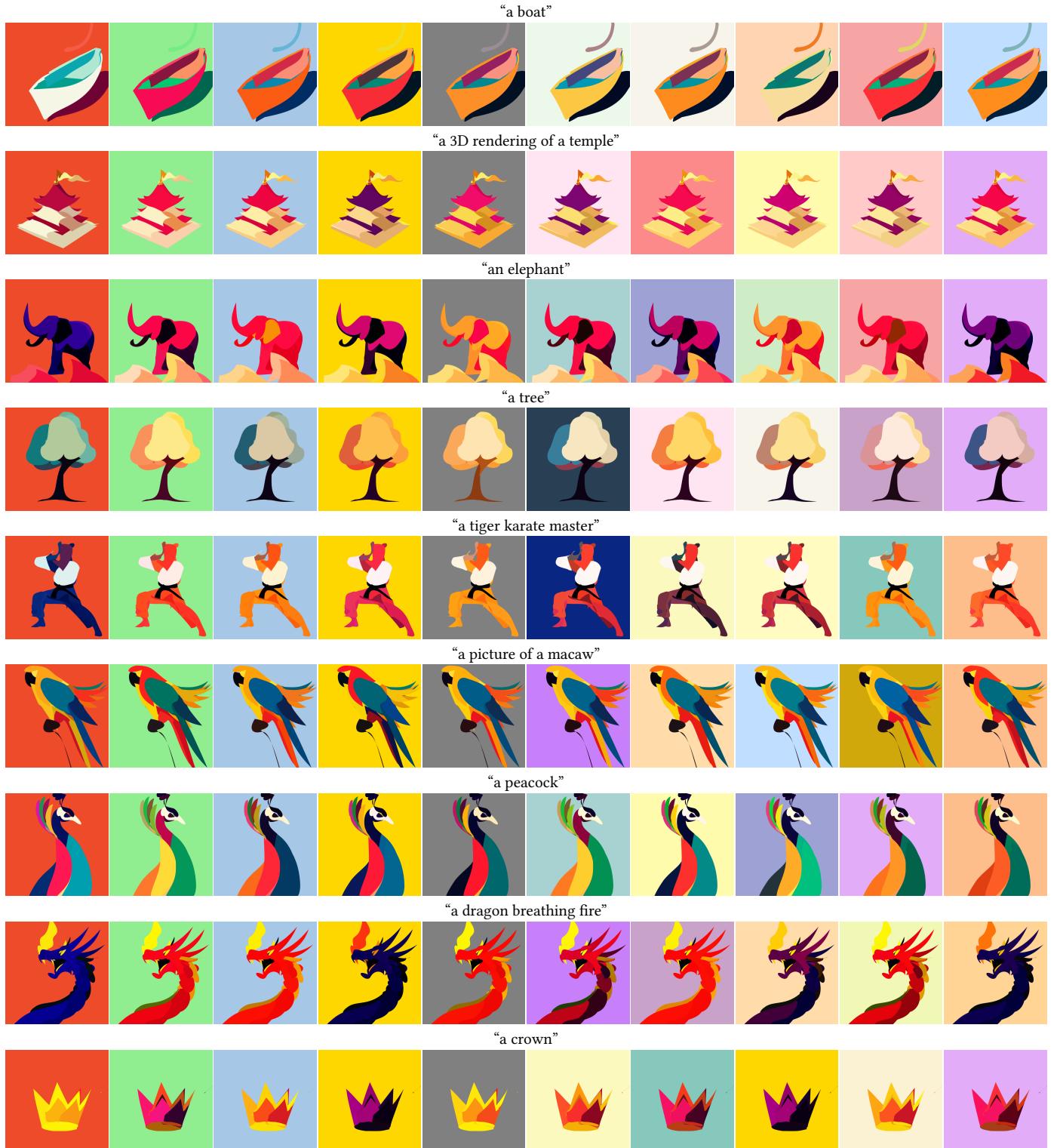
**Fig. 19. Shape Outlines of the Generated SVGs.** We present the corresponding outlines of SVGs generated by NeuralSVG, VectorFusion, and SVGDreamer for the results shown in Figure 17. The alternative methods often produce nearly pixel-like shapes that are difficult to modify manually. In contrast, NeuralSVG generates cleaner SVGs, making them more editable and practical.



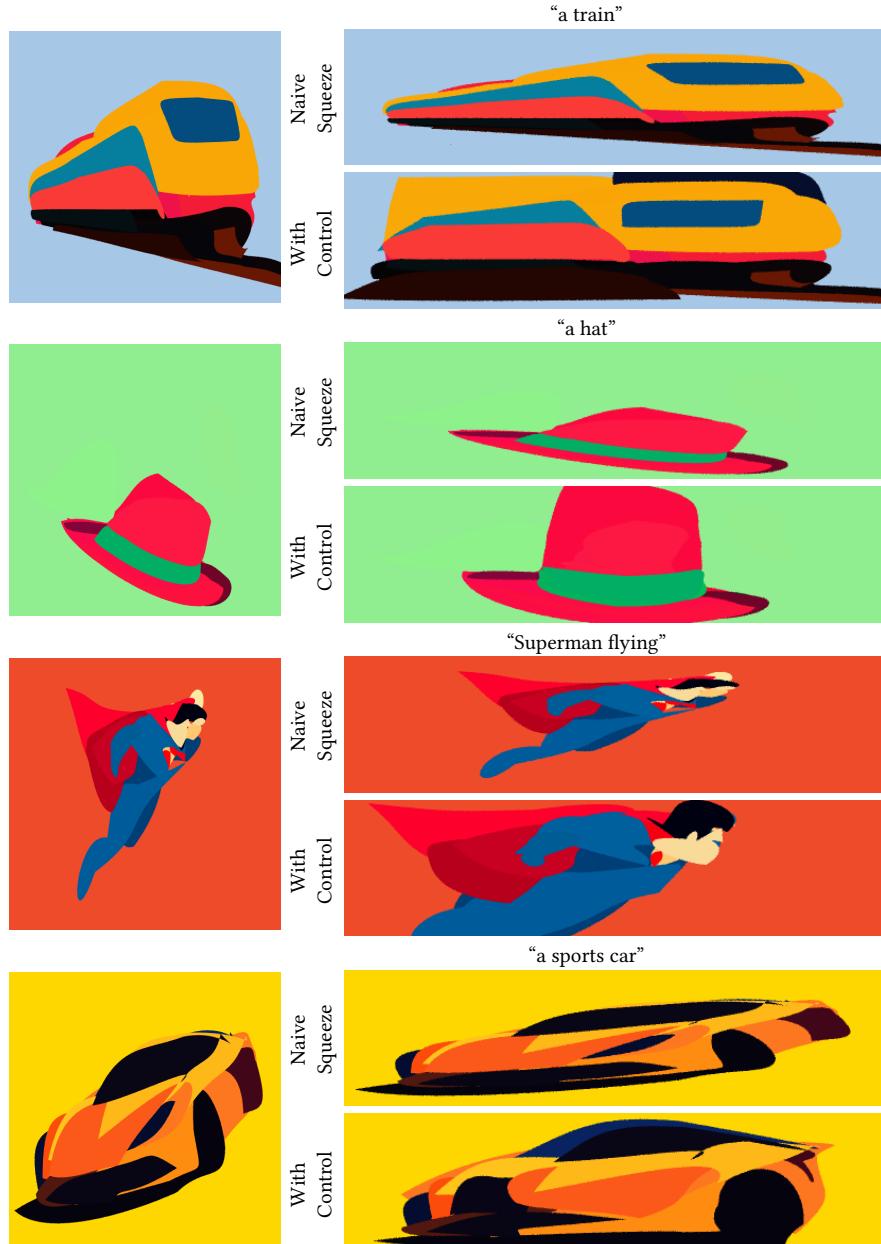
Fig. 20. **Qualitative Comparisons.** As no code implementations are available, we provide visual comparisons to NiVeL [Thamizharasan et al. 2024] (left columns) and Text-to-Vector [Zhang et al. 2024] (right columns) using results shown in their paper.



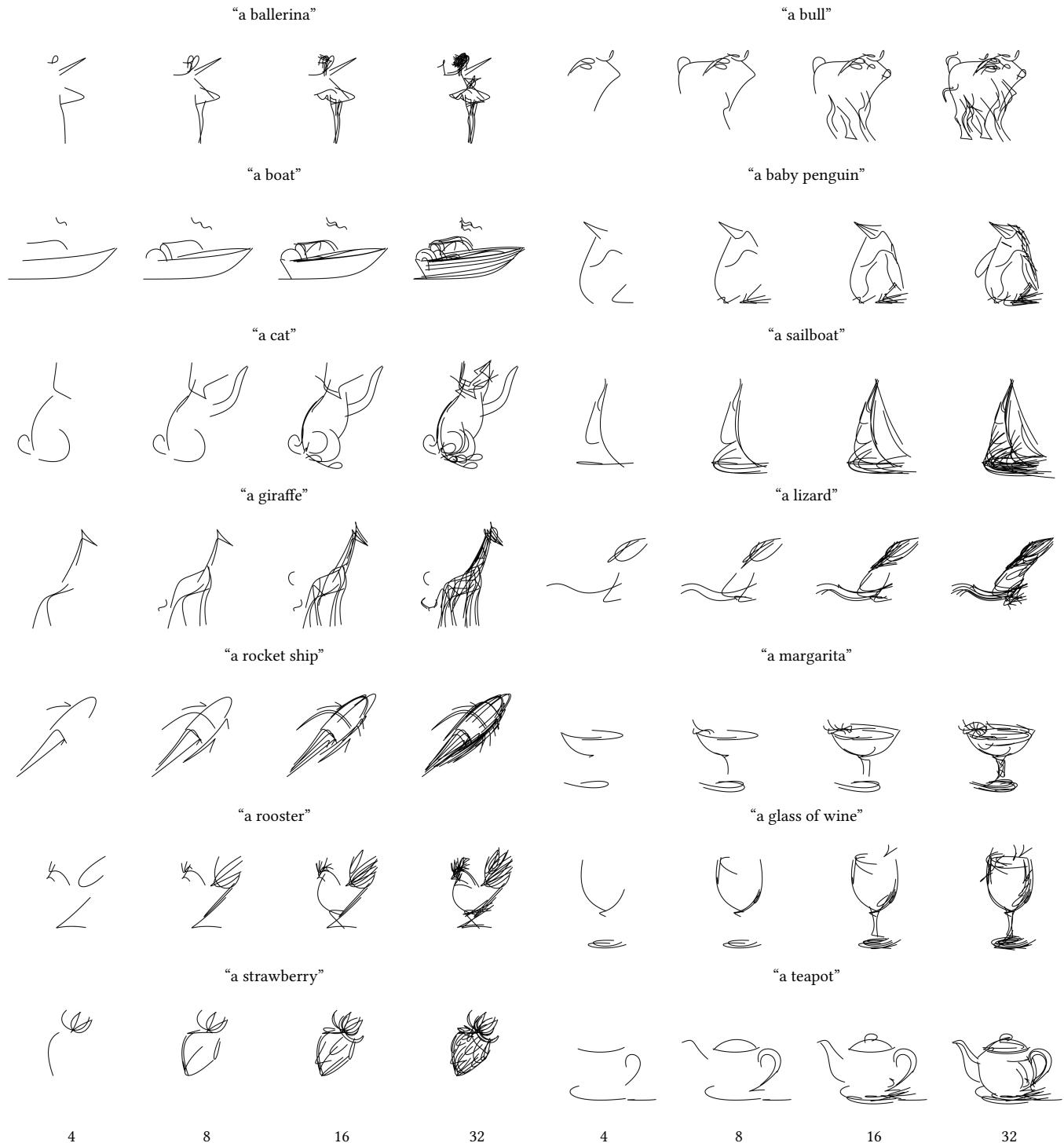
**Fig. 21. Dynamically Controlling the Color Palette.** Given a learned representation, we render the result using different background colors specified by the user, resulting in varying color palettes in the resulting SVGs. The 5 leftmost columns show colors observed during training while the 5 rightmost columns show unobserved colors.



**Fig. 22. Dynamically Controlling the Color Palette.** Given a learned representation, we render the result using different background colors specified by the user, resulting in varying color palettes in the resulting SVGs. The 5 leftmost columns show colors observed during training while the 5 rightmost columns show unobserved colors.



**Fig. 23. Dynamically Controlling the Aspect Ratio.** Additional results from optimizing NeuralSVG with aspect ratios of 1:1 and 4:1. In each pair of results, the top row shows the naive approach of squeezing the 1:1 output into a 4:1 aspect ratio. The bottom row shows the results where our trained network directly outputs the 4:1 aspect ratio.



**Fig. 24. Additional Sketch Generation Results.** NeuralSVG can generate sketches with varying numbers of strokes using a single network, without requiring modifications to our framework.